MACRO-BASED DYNAMIC DISCOVERY OF DATA SHAPE

FIELD OF THE INVENTION

[0001] The invention relates to the field of computer databases. In particular, the invention relates to an apparatus and method for simplifying database query constructs and enabling a computer program to dynamically discover the order and type of data requested through the simplified query.

BACKGROUND OF THE INVENTION

[0002] The invention described in detail below will be understood best in light of the following discussion of the state of the art and the problem that the invention solves.

[0003] Electronic databases provide a solution for long-term storage needs, as well as rapid search and retrieval of information. A database is merely a collection of data stored together and organized for rapid search and retrieval. In general, a database stores information internally as one or more "tables." A table is a collection of "records" (also called "rows"), and a record is a collection of "fields" (also called "columns"). Every record in a given table must have the same number of fields. Fields contain discrete data values.

[0004] In general, most databases are either hierarchical or relational. A database that implements a hierarchical data model links data together by embedding pointers within the data. The links in a hierarchical database are static. Static links decrease the complexity of data access, but limit flexibility. The relational model evolved as a means to improve the flexibility of data access. Relational models do not rely on static links. Rather, relational models allow a user to link data together dynamically. Dynamic links, in turn, enable a user to tailor data access to specific needs.

[0005] Typically, a database management system (DBMS) provides a user interface (UI) through which a user may access the information contained in a database. A DBMS commonly implements some form of command language that allows a user to construct a request for specific information. The Structured Query Language (SQL) is one example of a command language that has gained widespread acceptance in the art. A command that requests information from a database is referred to generally as a "query." A DBMS responds to a user's query by returning the requested information in a "result set." A result set may be displayed on the user's screen as a series of rows and columns, or may be saved to a file for future use or further processing.

[0006] Similarly, a DBMS usually provides an application program interface (API) through which other computer programs may access the information contained in a database. Although the implementations vary from one DBMS to another, many DBMS APIs make use of a command language similar to the command language that the DBMS implements in the UI. APIs commonly allow a programmer simply to embed the same command language within the program. And just as a DBMS returns a result set to a user in response to a user's query, a DBMS returns a result set to a program in response to the program's query.

[0007] A query generally consists of a statement that identifies what information should be retrieved and where that information can be found. Most DBMSs require a query to state (at a minimum) from which table or tables to retrieve the data (a "from-clause"), which fields to retrieve (a "select-clause"), and the selection criteria (a "where-clause"). If the query identifies two or more tables in a relational database, the query must also state the relation between the tables.

[0008] Regardless, though, of whether a DBMS is responding to a user's query or a program's query, the query statement determines the arrangement (or "shape") of the result set returned by a DBMS. In particular, the result set presents the fields in the order that they were stated in the select-clause. A user, of course, usually knows the order in which the fields were stated in the select-clause and, thus, knows the shape of the result set in advance. Similarly, a programmer that embeds query statements in a program knows the shape of the result set in advance. Advance knowledge of the result set's shape allows a programmer to build a program that anticipates a particular shape and processes the result set accordingly.

[0009] There are many occasions, however, when it is desirable to build programs that can interact with a database without knowing the shape of a result set in advance. For example, many programs are built to provide a layer of abstraction between a database and an end-user program. Programs that provide this layer of abstraction are commonly referred to as "middleware." In order to maximize usability and flexibility, a middleware program needs to be able to accommodate queries that come from an end-user program, or even end-users themselves. Thus, a middleware programmer usually will not know the result shape in advance, and must be able to discover dynamically the result shape as the program is executing. One possible method of dynamic shape discovery is to parse the query statement within the middleware program. Parsing a query statement, though, is a complicated process and requires substantial additional programming effort. Furthermore, the parsing method would result in slower execution and redundant parsing, since a DBMS must also parse the query statement. Therefore, a need exists for a method of dynamically discovering the shape of a result set, without the development and use of a complicated and time consuming language parser.

SUMMARY OF THE INVENTION

[0010] The present invention comprises a Dynamic Query Interface (DQI). The DQI provides an apparatus and method for dynamically discovering the shape of data returned by a DBMS in response to a query statement. The DQI comprises a Query Schema, a Select Macro, and a Macro Expansion Module (MEM).

[0011] A Query Schema comprises information about particular Query Entities and, if needed, Schema Relations. In particular, a Query Schema includes the names and types of fields that comprise each Query Entity.

[0012] A Select Macro comprises a query statement written in terms of the Query Entities and Schema Relations. Unlike conventional query statements, a Select Macro only requires a select-clause. The MEM builds a conventional query statement from a Select Macro. The MEM first breaks a Select Macro into Macro Tokens, and then compares each Macro Token with each Query Entity in the Query Schema. Then, if a Macro Token matches a Query Entity, the MEM expands the Macro Token to include the Query Entity fields designated in the Query Schema. The MEM also adds the matching Query Entity to the Select Macro as a from-clause and creates an appropriate join-clause based on Schema Relations.

[0013] The Macro Token expansion approach implemented in the MEM obviates the need for full language parsing and is thus much more suited to runtime execution than prior art solutions. Furthermore, because the MEM examines a Select Macro at runtime, the MEM has the added advantage over prior art solutions of being able to discover dynamically the shape of the requested data at runtime.

BRIEF DESCRIPTION OF DRAWINGS

[0014] **FIG. 1** is a depiction of a typical networked computing environment in which the integrated server architecture could be implemented;

[0015] **FIG. 2** represents the memory configuration of a typical computing workstation using the integrated server architecture;

[0016] **FIG. 3** illustrates the model for a Query Schema;

[0017] **FIG. 4** provides an example of a Query Schema for two tables; and

[0018] **FIG. 5** illustrates the operation of the Macro Expansion Module on a Select Macro in the context of the Query Schema in FIG. 4.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0019] The foregoing and other objects, features, and advantages of the invention will be apparent from the following more particular description of the preferred embodiment of the invention, as illustrated in the accompanying drawings wherein like reference numbers represent like parts of the invention.

[0020] As used herein, the term "database" means any collection of data stored together and organized for rapid search and retrieval, including without limitation flat file databases, fielded databases, full-text databases, object-oriented databases, and relational databases.

[0021] The term "from-clause" refers to a clause within a query that defines the data source or sources from which data should be retrieved.

[0022] The term "join-clause" refers to a clause within a query that defines one or more relationships between two or more data sources.

[0023] The term "query" refers to any statement that a processor recognizes as an instruction to retrieve data from a database.

[0024] The term "result set" means any collection of data retrieved from a database as a collection of records.

[0025] The term "result shape" means the order of fields within a result set.

[0026] The term "select-clause" refers to a clause within a query that defines which data should be retrieved.

[0027] As a person of skill in the art will appreciate, the DQI may be implemented in many different configurations, including software, hardware, or any combination thereof. For the sake of clarity and simplicity, the following discussion uses SQL to illustrate the operation of the preferred embodiment of the DQI. The use of SQL in the following discussion is not intended as a limitation of the present invention. SQL is a standardized query language well known to a person skilled in the art, and the syntax of SQL need not be described in detail here. A person of skill in the art will appreciate that any query language can be used with the present invention.

[0028] FIG. 1 is an illustration of computer network **100** associated with the present invention. Computer network **100** comprises local workstation **108** electrically coupled to network connection **102**. Local workstation **108** is electrically coupled to remote workstation **110** and remote workstation **112** via network connection **102**. Local workstation **108** is also electrically coupled to server **104** and persistent storage **106** via network connection **102**. Network connection **102** may be a simplified local area network (LAN) or may be a larger network such as a wide area network (WAN) or the Internet. Furthermore, computer network **100** depicted in FIG. 1 is intended as a representation of a possible operating network that may contain the present invention and is not meant as an architectural limitation.

[0029] The internal configuration of a computer, including connection and orientation of the processor, memory, and input/output devices, is well known in the art. The present invention is a methodology that can be embodied in a computer program. Referring to FIG. 2, the methodology of the present invention is implemented in DQI **220**, which resides in memory **200**. DQI **220** comprises Query Schema **222**, one or more Select Macros **224**, and Macro Expansion Module (MEM) **226**. DQI **220**, including Query Schema **222**, Select Macro **224**, and MEM **226** described herein can be stored within memory **200** of any workstation or server depicted in FIG. 2. Alternatively, DQI **220**, including Query Schema **222**, Select Macro **224**, and MEM **226** can be stored in an external storage device such as persistent storage **106**, or a removable disk such as a CD-ROM (not pictured). Memory **200** is only illustrative of memory within one of the machines depicted in FIG. 2 and is not meant as a limitation. Memory **200** also contains resource data **210**. The present invention may interface with resource data **210** through memory **200**.

[0030] In alternative embodiments, DQI **220** and/or any of the gateways can be stored in the memory of other computers. Storing DQI **222** and/or gateways in the memory of other computers allows the processor workload to be distributed across a plurality of processors instead of a single processor. Further configurations of DQI **220** across various multiple memories and processors are known by persons skilled in the art.

[0031] FIG. 3 illustrates the model for Query Schema **222**. In the preferred embodiment, as shown in FIG. 3, Query Schema **222** is itself implemented as a relational database. Query Schema **222** is comprised of an "Entity" table **302**, a "Field" table **304**, and a "Relationship" table **306**. Entity table **302** may contain many records for Query Entities. A Query Entity is any data structure composed of records, where each record contains one or more fields. In general, a

Query Entity corresponds to a database table, but may also be any other collection of data that can be represented as columns and rows, such as a database view. Every Query Entity in Entity table **302** should have one or more corresponding records in Field table **304**. Field table **304** contains a record for each field to be queried. Each record in Field table **304** should include the name of the field, but may also include additional information such as the type of data value bound to the field. Relationship table **306** contains a record for each pair of fields that creates a relationship between two or more Query Entities.

[0032] FIG. 4 provides an example of Query Schema **222** for two tables. The example in FIG. 4 illustrates a Query Schema **222** for a typical business database consisting of employee information. For this example, the hypothetical database contains at least two tables. The first table is named "Departments" and contains information about all the departments within the business. The second table is named "Employees" and consists of information about employees, including which department the employee works in. The database itself could contain many additional tables, but tables do not need to be included in Query Schema **222** unless they will be referenced in a Select Macro. Thus, for the sake of clarity and simplicity, this example will be limited to the Departments and Employees tables of the hypothetical database. Referring to FIG. 4, Entity table **302** in example Query Schema **222** consists of two entries corresponding to the two tables just described. Field table **304** contains an entry for each field in each table listed in Entity table **302** that might be used in a Select Macro. In FIG. 4, Field table **304** contains two entries for fields in the Departments table. The Departments fields are named "ID" and "Name." Similarly, Field table **304** also includes four entries for the Employees table. The Employee fields used in this example are named "ID," "Name," "Address," and "DepartmentID." Note that in this example, Field table **304** also includes the type of data value that each respective field

holds (i.e. "Int" for an integer value, or "Text" for character values). Relationship table **306** in FIG. 4 indicates that there is a single Entity Relation in example Query Schema **222**. The "Parent" field and the "Child" field in Relationship table **306** refer to specific (but different) IDs in Field table **304**. In this example, the single record in Relationship table **306** indicates that the ID field in the Departments table is related to the DepartmentID field in the Employees table. In particular, Relationship table **306** indicates that DepartmentID is dependent upon ID as a parent. In the context of a relational database, ID would be a "primary key" and DepartmentID would be a "foreign key."

[0033] A Select Macro comprises a query statement written in terms of Query Entities and Schema Relations. Unlike conventional query statements, though, a Select Macro only requires a select-clause. Thus, a simple Select Macro may consist of just two words – the "select" keyword and a Query Entity. For example, using the hypothetical database and example Query Schema **222** illustrated in FIG. 4 and discussed above, a simple two-word Select Macro would read:

select Departments;

Of course, any and all Query Entities in Query Schema **222** may be referenced. Thus, a Select Macro that references all the tables in Query Schema **222** would read:

select Departments, Employees;

Finally, a Select Macro may also contain a where-clause that restricts the results. Expanding upon the previous example, a Select Macro that restricts the results to a single employee would read:

select Departments, Employees where Departments.ID = 555;

A person of ordinary skill in the art will appreciate that a where-clause may be as complex as needed without affecting the shape of the result. Thus, a Select Macro may contain a complex where-clause originally developed for other queries, thereby decreasing development time and expense. FIG. 5 illustrates the operation of MEM **226** on this last example, Select Macro **224**, in the context of example Query Schema **222** discussed above. MEM **226** begins by reading Select Macro **224** and breaking it down into Macro Tokens (**501**). Here, the Macro Tokens would consist of 'select', 'Departments', 'Employees', 'where,' and 'Departments.ID=555'. MEM **226** then compares each Macro Token with each Query Entity in Entity table **302** (**505**). If MEM **226** matches a Macro Token with a Query Entity, then MEM **226** examines Field table **304**, retrieves all fields associated with the matched Query Entity (**507**), and adds each field to the select-clause (**509**). Thus, after MEM **226** matches the 'Departments' and 'Employees' Macro Tokens with their respective Query Entities, example Select Macro **224** would read:

> *select Departments.ID, Departments.Name, Employees.ID, Employees.Name, Employees.Address, Employees.DepartmentID where Department.ID=555;*

MEM **226** also creates a from-clause and appends each matched Query Entity (**511**). Thus, example Select Macro **224** becomes:

> *select Departments.ID, Departments.Name, Employees.ID, Employees.Name, Employees.Address, Employees.DepartmentID from Departments, Employees where Department.ID=555;*

Finally, MEM **226** examines Relationship table **306** to check for Schema Relations (**513**). As discussed above, Relationship table **306** in this example indicates that Departments.ID and Employees.DepartmentID are related. Specifically, Departments.ID is a parent key and Employees.DepartmentID is a child key in the relationship. Since both keys belong to Query

Entities that have been referenced in Select Macro **224** (i.e. Departments and Employees), MEM **226** inserts an inner join based on the entry in Relationship table **306** (**515**). Thus, the final version of the Select Macro that is forwarded to a DBMS reads as follows:

select Departments.ID, Departments.Name, Employees.ID, Employees.Name, Employees.Address, Employees.DepartmentID from Departments inner join Employees on Departments.ID=Employees.DepartmentID where Department.ID=555;

[0034] Referring again to FIG. 5 and the preceding discussion, a person of skill in the art will appreciate that MEM **226** can retain the number and order of fields added to the select clause as the fields are added. Thus, MEM **226** can dynamically discover the result shape of a Select Macro at runtime with little or no additional processing. Dynamic discovery of the result shape also enables MEM **226** to create internal data structures dynamically to retain the results for further processing. In the preferred embodiment, MEM **226** uses data objects to retain result sets. Data objects and object-oriented techniques are well known in the art and need not be described in detail here. MEM **226** creates a data object for each Query Entity in the select-clause. Thus, in the above example, MEM **226** would create a Department object having ID and Name attributes, and an Employee object having ID, Name, Address, and DepartmentID attributes. Furthermore, just as MEM **226** creates a join-clause based on Relationship table **306**, MEM **226** can also create a link or reference between objects based on Relationship table **306**. In another embodiment, MEM **226** uses an array or arrays to retain the result set. The advantages of dynamic shape discovery, though, are not limited to creating data objects or arrays, and a person of ordinary skill in the art will appreciate the many varied applications of DQI **220**.

[0035] Furthermore, a person skilled in the art will appreciate from the preceding discussion that various modifications and changes may be made to the preferred embodiment of the present invention without departing from its true spirit. This description is intended to be illustrative only and should not be construed in a limiting sense. The scope of the invention should be limited only by the language of the following claims.